JOURNAL OF SOFTWARE MAINTENANCE AND EVOLUTION: RESEARCH AND PRACTICE
J. Softw. Maint. Evol.: Res. Pract. 2007; **19**:361–382
Published online 21 June 2007 in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/smr.348

Research

Software evolution in open source projects—a large-scale investigation

Stefan Koch*, †

Institute for Information Business, Vienna University of Economics and BA, Augasse 2-6, Vienna A-1090, Austria

SUMMARY

In this paper, the evolution of a large sample of open source software systems will be analysed. The evolution of commercial systems has been an issue that has long been a centre of research, thus a coherent theoretical framework of software evolution has been developed and empirically tested, most notably the laws of software evolution. In exploring the evolutionary behaviour of open source systems, these results can serve as a point of reference, allowing to assess if differences exist, or which aspects of open and collaborative development styles have an impact on evolutionary behaviour. The data collection method relying on a large software repository and the respective source code control systems is described, and an overview on the collected data on several thousand projects is given. The evolutionary behaviour is explored using both a linear and a quadratic model, with the quadratic model being shown as better suited. The most interesting fact is that while in the mean the growth rate is linear or decreasing over time according to the laws of software evolution, a significant percentage of projects is able to sustain super-linear growth. There is a positive relationship between the size of a project, the number of participants, and the inequality in the distribution of work within the development team with the presence of super-linear growth patterns. On the other hand, there is evidence for a group of projects of moderate size which shows decreasing growth rates, while small projects in general exhibit linear growth. Copyright © 2007 John Wiley & Sons, Ltd.

Received 23 January 2006; Revised 23 April 2007; Accepted 4 May 2007

KEY WORDS: open source; software evolution; empirical study; software repository mining

1. INTRODUCTION

In the last years, free and open source software has gathered increasing interest, both from the business and academic world. As some projects in different application domains like Linux together

*Correspondence to: Stefan Koch, Institute for Information Business, Vienna University of Economics and BA, Augasse 2-6, Vienna A-1090, Austria.
†E-mail: stefan.koch@wu-wien.ac.at

with the suite of GNU utilities, GNOME, KDE, Apache, sendmail, bind, and several programming languages have achieved huge success in their respective markets, new business models have been developed and tested by businesses both small and large like Netscape or IBM. Academic interest into this new form of collaborative software development has arisen from very different backgrounds including software engineering, sociology, management or psychology, and has gained increasing prominence.

While many of the successful projects mentioned above are well known and produce output of high quality, a general assessment of the open source software development paradigm is yet outstanding. Currently, any discussion of this new model is mostly based on a small number of glamorous and successful projects, but these might constitute exceptions. Also several aspects of development processes can vary between different projects. Therefore, any analysis needs to be based on quantitative information on the enactment in a variety of project forms and sizes.

The main ideas of the open source development model are described in the seminal work of Raymond, 'The Cathedral and the Bazaar', in which he contrasts the traditional type of software development of a few people planning a cathedral in splendid isolation with the new collaborative bazaar form of open source software development [1]. In this, a large number of developer-turned users come together without monetary compensation to cooperate under a model of rigorous peer review and take advantage of parallel debugging that leads to innovation and rapid advancement in developing and evolving software products. In order to allow for this to happen and to minimize duplicated work, the source code of the software needs to be accessible, and new versions need to be released often. To this end, software licences that grant the necessary rights to the users, like free redistribution, inclusion of the source code, the possibility for modifications and derived works and some others have been developed. One model for such licences is the Open Source Definition, which lists a number of requirements for specific licences [2]. The most prominent example that fulfils these criteria while still being even more stringent, is the GNU General Public Licence, developed by the GNU project and advocated by the Free Software Foundation [3]. While there is no defined open source development model, but a list of general principles which projects follow completely or to some degree, open source projects in general share some characteristics like an open and collaborative development style.

The advantages and disadvantages of this new development model have been hotly debated [4,5], but mostly on a general level without empirical backing. While some detailed research has been undertaken to uncover issues like the organization of work in a small number of open source projects [6–9] and some aspects have been analysed using larger samples [10–15], several facets of the development process, including the expended effort and its estimation or the evolution of open source systems remain still largely unexplored on a large scale.

In this paper, the evolution of a large sample of open source software systems will be analysed. The evolution of commercial systems has been an issue that has long been a centre of research, thus a coherent theoretical framework has been developed and empirically tested, most notably the laws of software evolution. In exploring the evolutionary behaviour of open source systems, these results can serve as a point of reference, allowing to assess if differences exist, which aspects of open and collaborative development styles have an impact on evolutionary behaviour, and analyse possible reasons for differences. In addition, modelling the growth rate in open source projects can be interesting for developing models to predict future evolution, maintainability and other characteristics, both on project and wider levels. Given the increasingly widespread commercial

DOI: 10.1002/smr

interest and investment into open source, such results would be of high interest to several parties. Lastly, open source provides a huge number of projects with publicly available data for software engineering studies in general, and software evolution in general. While until today much empirical research is based on small numbers of projects and often unavailable raw data due to commercial interests, studies today could, if any differences found are accounted for, include thousands of projects and be completely replicable.

In the next section, a literature review both on software evolution in general and prior research on evolution of open source projects will be given. After that, the method used for data collection will be described together with an overview of the data set. Then the software evolution in this sample will be analysed. The paper ends with a discussion of threats to validity and a conclusion.

2. LITERATURE REVIEW

The study of software evolution for commercial systems was pioneered by the work of Lehman and Belady on the releases of the IBM OS/360 operating system [16], which has led to many other works, also based on other software systems [17,18], in which the laws of software evolution were formulated, expanded, and revised. These laws entail a continual need for adaptation of a system. This continual adaptation leads to increased complexity of the system. As it is assumed that constant incremental effort is applied at each time step, average incremental growth naturally declines. Turski [19] for example has modelled this as an inverse square growth rate, others use a linear model. During the years, several other empirical studies have been conducted and published on software evolution in non-open source software systems, Kemerer and Slaughter [20] and Scacchi [21] provide overviews of such works.

For the area of open source software systems, several works have explicitly dealt with the topic of software evolution. The first and one of the most important contributions is a case study by Godfrey and Tu [22], who have analysed the most prominent example available, the Linux operating system kernel. Using the size in lines-of-code (LOC) as a function of the time in days since release 1.0, they found that the growth behaviour is best fitted by a super-linear rate. This result significantly contradicts the prior theory of software evolution, which postulates a decline in growth best modelled either using a linear or an inverse square rate. This would, therefore, give an indication of major differences in development models and their results.

On the other hand, Paulson *et al*. [23] have used a linear approximation, and have not found any differences in growth behaviour between open and closed source software projects. In their study, they analysed three widely known open source projects (Linux, Apache, and GCC) and three closed source systems.

Robles *et al*. [24] reproduced the study of Godfrey and Tu [22] with newer data, and found similar results, in addition showing that the growth of Linux has even accelerated during the five years between both works. They have also analysed major subsystems, finding also super-linear growth patterns on this level. To validate these results, the authors have compared this to the family of *BSD kernels, which in contrast show an almost linear growth pattern with the exception of FreeBSD until the year 2000, where super-linear growth is present. In addition, 18 more large open source projects (like Apache, GNOME, or KDE) were analysed, finding growth patterns linear or

DOI: 10.1002/smr

close to linear for 16 of them, with the other two exhibiting some special characteristics. From the fact that the studied systems show a growth rate higher than a smooth one, the authors conclude that the fourth law of software evolution, 'conservation of organizational stability' which implies constant incremental effort, possibly does not apply to these large open source projects. In another case study, Robles *et al*. [25] found that the KDE project shows super-linear growth.

Capiluppi *et al*. [26] presented the first large horizontal study of open source system evolution using 406 projects from a repository, focusing more closely on 12 'alive' projects out of this set. In the full data set, the authors observe that over six months, 97% of projects did not change size or changed less than 1%. In the sample of alive projects, size is constantly growing, from which the applicability of the laws of software evolution is hypothesized, but no model is fitted to the data to further explore this notion. They also note that in large and medium projects, the number of modules grows, but their size tends to evolve to a stable value.

Robles-Martinez *et al*. [27] have applied a similar methodology to this work based on the use of publicly available data in the form of the Concurrent Versions System (CVS) [28] source code control system, and have detailed the evolution of MONO as a case study from several perspectives, including commits, authorship and also size in LOC. They conclude that the evolution of the studied modules proceeds with rather different growth rates, but did not fit any model to the data.

Nakakoji *et al.* [29] have also studied the evolution of open source software systems, but take a broader perspective in also examining the evolution of the associated communities and the relationship between both types. Using four case studies, they link the system evolution on the level of different versions and branches to the community evolution, and arrive at a classification with three different types of projects.

Scacchi [30] gives an excellent discussion of open source software evolution, with an overview and review of studies both on proprietary and open source projects. He concludes from this analysis that the laws of software evolution as presently stated and based primarily on the study of large closed source systems do not account for the potential for super-linear growth in software size that can be sustained by satisfied developer–user communities. He also stresses the importance of the availability of data on open source projects for further studies on software evolution and software engineering in general.

3. METHOD AND DATA SET

3.1. Data collection

For performing the proposed analysis of the evolutionary behaviour of open source software projects, the information contained in software development repositories will be explored. These repositories contain a plethora of information on the underlying software and the associated development processes [31,32]. Studying software systems and development processes using these sources of data offers several advantages [31]: this approach is very cost-effective, as no additional instrumentation is necessary, and it does not influence the software process under consideration. In addition, longitudinal data are available, allowing for analyses considering the whole project history. Depending on the tools used in a project, possible repositories available for analysis include source code versioning systems, bug reporting systems, or mailing lists. Many of these have already been

DOI: 10.1002/smr

used as information sources for closed source software development projects. For example, Cook *et al.* [31] present a case study to illustrate their proposed methodology of analysing in-place software processes. They describe an update process for large telecommunications software, analysing several instances of this process using event data from customer request database, source code control, modification request tracking database, and inspection information database. Atkins *et al.* [32] use data from a version control system in order to quantify the impact of a software tool, a version-sensitive editor, on developer effort. Kemerer and Slaughter [20] in their study on software evolution use a coding of change events found in histories or logs written by maintenance programmers.

In open source software development projects, repositories in several forms are also in use, in fact form the most important communication and coordination channels, as the participants in any project are not collocated. Therefore, only a small amount of information cannot be captured by repository analyses because it is transmitted inter-personally. As a side effect, the repositories in use must be available openly and publicly, in order to enable as many persons as possible to access them and to participate in the project. Therefore, open source software development repositories form an optimal data source for studying the associated type of software development.

Given this situation, repository data have already been used in research on open source software development. This includes in-depth analyses of small numbers of successful projects like Apache and Mozilla [6,7], GNOME [8], or FreeBSD [9] using mostly information provided by version control systems, but sometimes in combination with other repository data like from mailing list archives. Large-scale quantitative investigations spanning several projects going into software development issues are not yet as common, and have mostly been limited to using aggregated data provided by software project repositories [10–12], meta-information included in Linux Software Map entries [13], or data retrieved directly from the source code itself [14].

In this paper, we will follow this approach of using publicly available data from software repositories to study the evolutionary behaviour of open source projects and the underlying software process.

For this analysis, a large data set covering a diverse population of projects was needed. SourceForge.net, the software development and hosting site, was chosen as the source of data. The mission of SourceForge.net is 'to enrich the open source community by providing a centralized place for open source developers to control and manage open source software development'. To fulfil this mission goal, a variety of services is offered to hosted projects, including tools for managing support, mailing lists and discussion forums, web server space, shell services and compile farm, and source code control. While SourceForge.net publishes several statistics, e.g., on activity in their hosted projects, this information was not detailed enough for the proposed analysis. For example, Crowston and Scozzi [10] used the available data for validating a theory for competency rallying, which suggests factors important for the success of a project. Hunt and Johnson [11] have analysed the number of downloads of projects occurring, and Krishnamurthy [12] used the available data of the 100 most active mature projects for an analysis.

As sources of data within SourceForge.net, especially the source code control system offered, in the form of CVS, a free system which is being used extensively in the free software community [28], was identified as being able to provide necessary information. Several works have already demonstrated that important information about software development can be retrieved from such repositories [6–9]. In addition, data from the web pages of the projects hosted was retrieved. The process employed for data retrieval is depicted in Figure 1. Fischer *et al.* [33] have employed a

DOI: 10.1002/smr

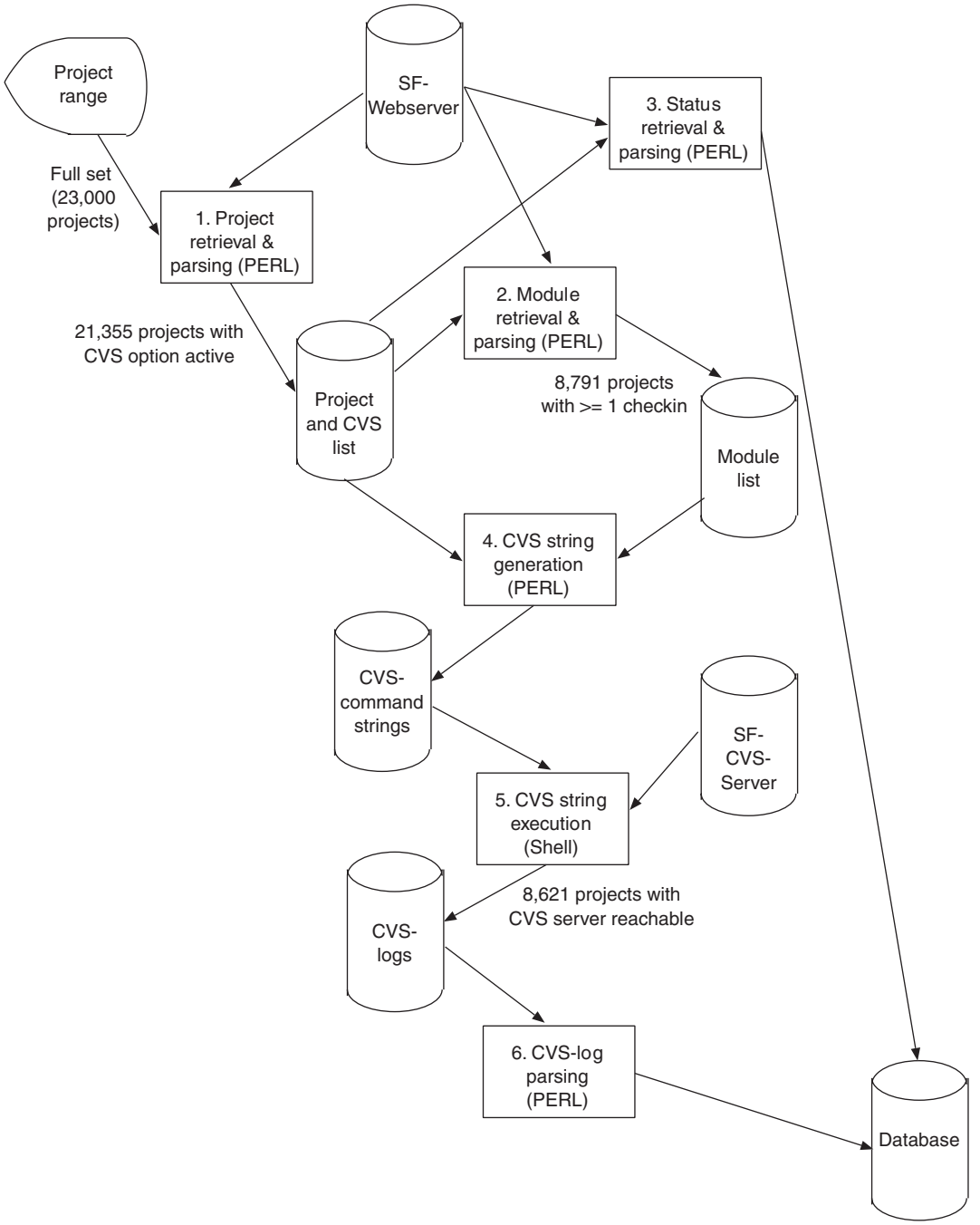

Figure 1. Data retrieval process.

DOI: 10.1002/smr

similar approach, but combining CVS data with bug tracking data and using the Mozilla project for evaluation.

As a first step, a data set had to be defined. In this case, all projects hosted and actively using CVS were selected to be included in the analysis. Therefore, the first step was to consult the SourceForge.net homepage that displays the number of currently hosted projects (at the relevant date 23 000). All possible project numbers starting by one and up to this number were selected. By querying for each project (as identified by its number) its CVS information web page hosted at SourceForge.net, a list of projects which are both still hosted and have the CVS service enabled was retrieved. This resulted in one HTML page retrieved per project which was then parsed for the necessary information regarding CVS server name and password. Both tasks were performed using Perl scripts (step 1), resulting in 21 355 candidate projects with enabled CVS service. The project titles, CVS server names, and passwords were extracted and stored. As SourceForge.net also has a development or maturity status indicator assigned to each project, this information was retrieved for the projects using again Perl scripts for downloading the relevant web pages (the summary page for each project) and parsing them (step 3). The resulting status was stored in the database for each project to be used later for analysis.

As the CVS interface can only handle statements concerning the modules of which a project is composed, the names of the modules of each project were also necessary. In addition, this would yield the information that projects actively use the CVS service. Therefore, the web page for browsing the CVS repository was retrieved for each project in the list and parsed (step 2). This showed that only 8791 projects were actively using the CVS service, i.e., during the observed time period between project start and data retrieval showed any commits, and thus were usable for further analysis. Using this information together with the CVS server name information, a Perl program was used to generate a shell script for querying the CVS servers and retrieving the necessary data (step 4). This was done by first checking out the source code for each module and then issuing the 'log' command for it (which is only possible for checked out items). Therefore, for each project a number of commands were necessary, including check out, log, creating a directory, and sleep periods.

Executing this large shell script (about 110 000 statements) resulted both in the downloaded source code and an output log file for each project (step 5). The CVS log command produces the whole history of all files in the module. This shows the work of the programmers on the project by submitting ('checking in', 'committing') files. A commit can pertain to several files, resulting in a revision for each of them. The CVS-repository records every revision with the changes in the LOC and further information. For a more in-depth description of CVS and available data, see also Fischer *et al*. [33]. This information, was then extracted from the logs by yet another Perl script and stored in the database (step 6). The number of LOC checked in with the first commit for each file ('initial revision') was computed from the source code itself, as it is not recorded automatically in CVS.

Based on this data collection, the following metrics are available: the first metric used is the number of LOC added to a file. The definition of this often disputed metric LOC [34] is taken from the CVS repository and, therefore, includes all types of LOC, e.g., also commentaries [28]. In addition, any LOC changed is counted as one LOC added and one LOC deleted. The LOC deleted are defined analogous. The difference between the LOC added and deleted, therefore, gives the change in size of a software artefact under consideration in the corresponding time period. These changes can be cumulated to give the size at any moment. As files, all types of

DOI: 10.1002/smr

files have been included, so there has not been a limitation to source code files. The metric of revision refers to the submission of a single file by a single programmer. The next metric was directly taken from the SourceForge.net repository, which has a development or maturity indicator assigned to each project. This indicator has seven possible values, reaching from planning, pre-alpha, alpha, beta to production/stable and mature, and to inactive. This indicator is assigned by the project administrator, and therefore need not necessarily be a correct description of the current status. Lastly, we define a programmer as being active in a given period of time if he performed at least one commit during this interval. For the following analyses, a time window of one month has been used. With the data available, programmers are identified by their CVS account, so everybody with a CVS account having performed at least one commit is defined as a programmer.

Subsequent analyses were performed using queries to the database and processing with R, a free statistics package. Overall, information was retrieved for 8621 projects, mostly due to some projects having disabled CVS between module identification (step 2) and actual download (step 5). The download took more than one month during the end of 2001, and the downloaded files use about 33 GB of disk space. All downloads and queries to the SourceForge.net servers were supplied with ample sleeping periods so as to not delay services for other users due to overloads. More details on the data retrieval can be found in Hahsler and Koch [35], additional analyses of these data can be found in Koch [15], while in the following a short overview is provided.

3.2. Data set

Within the complete set of projects, a total of 7 734 082 revisions have been made, with 663 801 121 LOCs having been added and 87 405 383 having been deleted. The projects consist of 2 474 175 single files, and an overall number of 12 395 distinct programmers have contributed with at least one commit to the CVS. The distribution of both the assets available, i.e., the programmers, and the resulting outcome, i.e., revisions, LOC and project status within the project ecology is very skewed. Table I gives descriptive statistics for the associated variables from this data set.

As can be seen, the vast majority of projects have only a very small number of programmers (67.5% have only one programmer), only 1.3% have more than 10 programmers. This number of programmers can be shown to follow a power law (or Pareto or Zipf) distribution [15], like Hunt and Johnson [11] have also found for the number of downloads of projects. These numbers also correspond to the findings of Krishnamurthy [12], who showed that most of the projects had only a small number of participants (median of 4). Only 19% had more than 10, 22% had only one developer. While this percentage is much smaller than found here, this is not surprising as Krishnamurthy used only the 100 most active projects.

Regarding the output of the projects, a similar situation can be seen, the vast majority of projects achieves only a small number of revisions and is of small size, leading to the assumption that input and output of projects are correlated, i.e., that projects with a small number of programmers only achieve small numbers of revisions and LOC. This intuitive relationship can be ascertained. For example, the total number of programmers of a project correlates positively at 1% significance with coefficients 0.472 with number of revisions and 0.408 with total LOC added [15].

Regarding the situation within projects, most prior studies [6–9,14,36] have found a distinctly skewed distribution of effort between the participants. Similar results can also be found at the

DOI: 10.1002/smr

Table I. Descriptive statistics for project variables from SourceForge.net data set ($N = 8621$).

	Definition	Min.	Max.	Mean	Std. dev.	Median
Number of programmers	Number of distinct CVS accounts having performed at least one revision for a project	1	88	1.86	2.61	1
Revisions	Number of submissions of a single file to a project	1	133 759	897.12	3840.90	192
LOC added	LOCs added to a project by all programmers based on CVS definition (e.g., including commentaries)	0	12 951 K	77 K	459 K	10 801
LOC deleted	LOCs deleted from a project by all programmers based on CVS definition (e.g., including commentaries)	0	3847 K	10 K	73 K	373
Files	Number of files of a project (including all types)	1	42 674	285.46	1317.74	69
Age	Number of months between first revision of a project and time of data retrieval	1	133	11.38	17.81	9
Development status	SourceForge.net maturity status indicator with seven possible values (planning, alpha, beta, etc.)	0	6	2.67	1.77	3

project ecology under consideration (see Table II for variables of programmer participation in SourceForge.net). The top decile is responsible for 79% of the total SourceForge.net code base, the second decile for additional 11%. In this study, we used a simple measure for the inequality of work distribution within the development team [15,37], computing for each programmer in the team the squared difference between the percentage of revisions he contributed, and the percentage he should contribute if everyone contributed the same amount, and summing up for all team members. An increasingly inequal distribution of work then shows rather small, but positive correlations with the total number of revisions and sum of LOC added. There is no correlation with the age of the project, so the inequality does not increase simply with time. Of course, the direction of the relationship is not ascertained, so the results do not necessarily indicate that more activity in projects is caused by a more unequal distribution of contributions, as the other way would also give a possible explanation, i.e., as the project grows, the inequality grows as a result.

The next possible influence on productivity in a project is the number of active programmers, following the reasoning behind Brooks's law [38], 'Adding manpower to a late project makes it later', that communication costs are increasing at a super-linear rate with the number of participants. Therefore, the number of active programmers and the achieved progress in each project was analysed on a monthly basis, using hypotheses $H1$.

Hypothesis *H*1. *A higher number of active programmers in a month has a negative influence on productivity as measured by the mean output per programmer in LOC in this month. Likewise, a higher number of active programmers in a month has a negative influence on productivity as measured by the mean output per programmer in revisions in this month.*

DOI: 10.1002/smr

Table II. Descriptive statistics for programmer variables from SourceForge.net data set ($N = 12\,395$).

	Definition	Min.	Max.	Mean	Std. dev.	Median
Revisions	Number of submissions of a single file performed	1	132 736	624	2815	112
LOC added	LOCs added to all project based on CVS definition (e.g., including commentaries)	0	16 152 866	53 554	374 750	5469
LOC deleted	LOCs deleted from all project based on CVS definition (e.g., including commentaries)	0	3 846 863	7052	54 547	372
Files	Number of files of all projects for which at least one revision was performed (including all types)	1	44 271	230	1105	48
Projects	Number of projects for which at least one revision was performed	1	15	1.29	0.81	1

Although the number of active programmers can be shown to have a significant and positive relationship with the overall output in this data set, the coefficient is much smaller than previously found by Koch and Schneider in their analysis of the GNOME project [8]. A productivity decrease due to more participants itself is next to non-existent, as the correlation between the number of active programmers and the mean output per programmer in a period is (although negative) only -0.013 with both revisions and LOC used as output measures (significant at 5%). Therefore, $H1$ is not confirmed, as the correlation is too small to give any significant effect.

4. ANALYSIS OF SOFTWARE EVOLUTION IN OPEN SOURCE SYSTEMS

4.1. Growth modelling

Using the data retrieved for the projects from the SourceForge.net project ecology as detailed above, the evolutionary behaviour of these systems is explored. The first idea is to analyse whether a linear or other growth pattern is present in the data. This is formulated as hypothesis $H2$, to be first tested on the full data set.

Hypothesis *H*2-fullset. *The growth of open source software systems in LOC is better modelled using a quadratic function of time than a linear one.*

To this end, both a linear and a quadratic model are computed for each project, taking the size in LOC S as a function of the time in days since the first commit t, which is used as project start date, and using one month as time window. This approach is similar to Godfrey and Tu [22], who used release 1.0 as starting point and further releases of Linux, respectively [24]. Therefore, model A was formulated simply as

$$S_A(t) = a * t + b \tag{1}$$

and model B as

$$S_B(t) = a * t^2 + t * b + c \tag{2}$$

DOI: 10.1002/smr

The necessary parameters were estimated using regression techniques. This was possible for 4047 of the projects, as the other lacked necessary information, i.e., had too few data points to be able to determine regression parameters with enough confidence (e.g., for projects with activity in only one month, no regression can be fitted). In order to compare the resulting fit of these models and thus test hypothesis $H1$, the adjusted R^2 measure is used which accounts for the number of parameters. Therefore, the fact that the quadratic model is more flexible, and also contains the linear model as a special case (for $a = 0$) is considered in the results. The quadratic model B outperformed the linear one with a mean adjusted R^2 of 0.831 (median 0.918) against 0.592 (median 0.727). This difference was tested for significance, using a Wilcoxon signed rank test instead of a paired-samples t-test because the difference scores are not normally distributed. Therefore, the associated null hypothesis $H1_0$-fullset that the distributions are equal is indeed rejected (at 1% significance). Nevertheless, we will revisit this sharp distinction later on in this paper.

These first results are not surprising and are still in line with the laws of software evolution. As a next step, it is therefore necessary to explore whether or not the growth rate is decreasing over time according to these laws. This can be checked by analysing the second derivative of the quadratic model $S_B(t)'$, or more conveniently directly the coefficient of the quadratic term a. As a first step, it is confirmed that the distribution of this parameter indeed is different from zero, using a t-test at 1% significance. Then, the distribution of this term is explored. In the mean, it has a slightly negative value of -0.504 with a median of -0.020. This would indeed indicate a decreasing growth rate in accordance with the laws of software evolution, but contrary to the findings of Godfrey and Tu [22] and Robles *et al.* [24] for Linux, respectively, Robles *et al.* [25] for KDE. In fact, for 61% of the projects this term is negative, for the remaining minority of 1578 it is positive. This shows that a rather large number of projects exhibit super-linear growth.

4.2. Project characteristics

As a next step, it is explored whether there are any characteristics of projects that lead to this super-linear growth behaviour. Therefore, the projects are divided into two groups according to their growth behaviour, i.e., whether it is super-linear or not. This is formulated as a series of hypotheses, to be tested first on the full set of projects, therefore denoted as follows.

Hypothesis *H*3-fullset. *Projects achieving super-linear growth are larger in number of LOC than those not.*

Hypothesis *H*4-fullset. *Projects achieving super-linear growth are more active in number of revisions than those not.*

Hypothesis *H*5-fullset. *Projects achieving super-linear growth have more programmers than those not.*

Hypothesis *H*6-fullset. *Projects achieving super-linear growth have a higher inequality than those not.*

For testing these hypotheses, Spearman correlation coefficients and non-parametric Mann–Whitney U-tests are employed due to the fact that all variables are not normally distributed. This

DOI: 10.1002/smr

can be verified by employing a Kolmogorov–Smirnov test, in all cases at 1% significance. For testing all hypotheses, the respective values at the time of last revision was used.

In this analysis, a small but significant (at level 0.01) relationship with size in LOC, activity in number of revisions and number of programmers, both at the time of data retrieval, can be found, indicating that larger projects with a higher number of participants might be more often able to sustain super-linear growth (see also Figure 2). This supports hypotheses $H3$-fullset, $H4$-fullset, and $H5$-fullset, and would be in accordance to the findings of Godfrey and Tu [22], respectively, Robles *et al*. [25], as both Linux and KDE are relatively large projects, but would contradict the assumption of software evolution that increased size leads to more complexity and interdependencies, thus decreasing growth rate. It would also be in conflict with the findings of Robles *et al*. [24], who have found linear growth rates in 16 out of 18 large open source projects.

As a further explanatory variable, the inequality of work distribution within the team was explored. Again, a small but significant positive relationship (significance level 0.05) was found, both when computing the inequality based on LOC and on revisions (see also Figure 3), supporting hypothesis $H6$-fullset.

This indicates that in the projects with super-linear growth rate, the distribution is in general more inequal. At first glance, this seems to contradict the prior result of these projects having more participants, but indeed seems to point at a certain development model. A project with super-linear growth has a higher number of participants, but this does not necessarily lead to a more equal distribution of output within this group. Indeed, it can verified that in the project population, there is a significant correlation with coefficients over 0.9 between inequality, based on LOC or revisions,

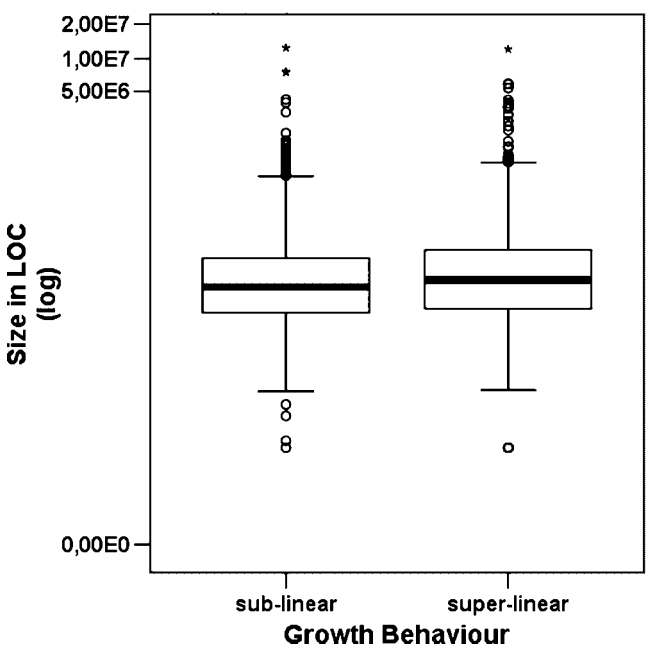

Figure 2. Boxplot of project size distribution depending on growth behaviour.

DOI: 10.1002/smr

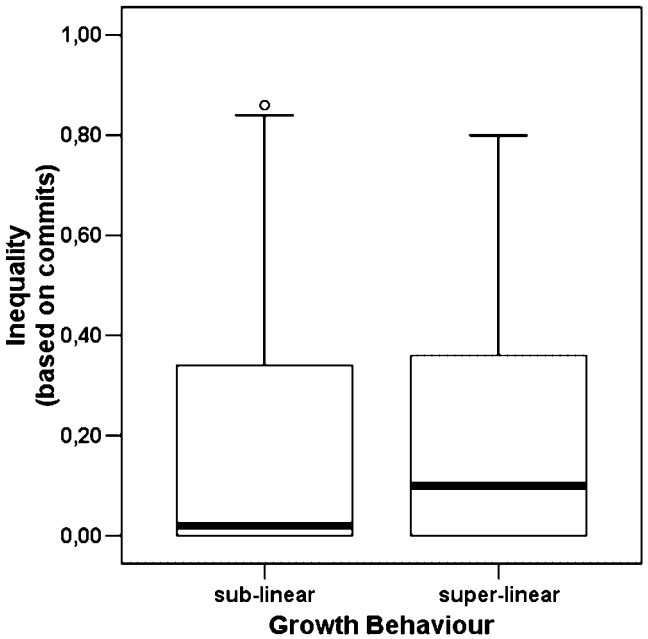

Figure 3. Boxplot of project inequality distribution depending on growth behaviour.

and number of participants (at level 0.01). Therefore, a few people within this larger group seem to do most of the work, while the others assist them in large numbers. This organisation has been proposed decades ago by Mills [39] as 'chief programmer team organization' [40], also termed 'surgical team' by Brooks [38], in which system development is divided into tasks each handled by a chief programmer who is responsible for the most part of the actual design and coding, supported by a larger number of other specialists like a documentation writer or a tester. This finding will need to be further explored, as this form of organization seems to be able to overcome the problems associated with increased complexity during software evolution.

4.3. Analysis of subsets

To check the validity of these results, we will explore two different subsets of the data set separately, again applying hypotheses $H2$–$H6$. First, only projects having achieved a given maturity (development status as assigned by the project administrator productive or mature) have been included in a separate analysis, which leaves about a quarter (1087 projects) of the data. In this data sample, the hypothesis that open source projects might experience super-linear in their early phases, but slow down later to form a S-curve [21], can be checked to some extent. Robles *et al*. [24], for example, did not find any difference in growth patterns before and after the first stable results. Using this subset, the results are verified, using respective hypotheses $H2$-mature to $H6$-mature. Model B employing a quadratic form is still a significantly better fit than the linear model A, support $H2$-mature. Analysing further the second derivative of this model, again the growth rate in the

DOI: 10.1002/smr

mean (−0.586) and median (−0.011) is decreasing, and the percentage of projects with super-linear growth is also nearly identical (425 out of 1087). Regarding project characteristics, the relationship between super-linearity and inequality is no longer significant, the other remain, supporting $H3$-mature, $H4$-mature and $H5$-mature, but not $H6$-mature.

As a second subset, only those projects have been selected which have achieved at least moderate size. This has been defined as having at least five distinct programmers and a size of at least 100 K LOC at the time of data retrieval. In this set, 207 projects are included, applying hypotheses $H2$-large to $H6$-large. Again, results confirm that the quadratic model outperforms the linear one and therefore $H2$-large, but the second derivative is now close to zero. A t test of the distribution on this value is no longer significant. The mean value is still negative with −0.364, but the median with 0.0023 is already positive with half of the projects now showing super-linear growth rate (104 projects). Regarding project characteristics, all relationships hold, although size and inequality only at 5% significance, supporting all hypotheses $H3$-large to $H6$-large.

4.4. Revised growth modelling

Lastly, the sharp distinction between two groups of projects, those experiencing super-linear growth and those that do not, is revisited. A new group is introduced representing linear growth in contrast to sub- and super-linear rates. This group is defined as those projects having either a better fit for the linear than the quadratic model, or a coefficient of the quadratic term between −0.1 and 0.1, thus being very near to zero. Using this definition, this new group becomes the largest, with 42.3%, followed by 35.7% with sub-linear and 22% with super-linear growth. Therefore, still a significant amount of projects exhibit super-linear growth. Again using Mann–Whitney U-tests, the relationships between group membership and project characteristics are explored, using the adapted hypotheses as follows.

Hypothesis $H3'$. *Projects achieving super-linear growth are larger in number of LOC than those showing linear growth behaviour, and projects achieving super-linear growth are larger in number of LOC than those showing sub-linear growth behaviour.*

Hypothesis $H3''$. *Projects achieving linear growth are larger in number of LOC than those showing sub-linear growth behaviour.*

Hypothesis $H4'$. *Projects achieving super-linear growth are more active in number of revisions than those showing linear growth behaviour, and projects achieving super-linear growth are more active in number of revisions than those showing sub-linear growth behaviour.*

Hypothesis $H4''$. *Projects achieving linear growth are more active in number of revisions than those showing sub-linear growth behaviour.*

Hypothesis $H5'$. *Projects achieving super-linear growth have more programmers than those showing linear growth behaviour, and projects achieving super-linear growth have more programmers than those showing sub-linear growth behaviour.*

Hypothesis $H5''$. *Projects achieving linear growth have more programmers than those showing sub-linear growth behaviour.*

DOI: 10.1002/smr

Hypothesis *H6′*. *Projects achieving super-linear growth have a higher inequality than those showing linear growth behaviour, and projects achieving super-linear growth have a higher inequality than those showing sub-linear growth behaviour.*

Hypothesis *H6″*. *Projects achieving linear growth have a higher inequality than those showing sub-linear growth behaviour.*

In this case, as two consecutive tests are employed in testing $H3'$, $H4'$, $H5'$, and $H6'$, Bonferroni correction is used to adapt the significance level for these tests accordingly. Nevertheless, the results obtained above are in general verified, with projects experiencing super-linear growth being larger, having more participants and a more inequal distribution of effort than those in the other two groups. Only the difference in inequality to the sub-linear growth rate group is not statistically significant. Thus, hypotheses $H3'$, $H4'$, and $H5'$ are unconditionally supported, $H6'$ to some extent. The interesting and new result is that, in all of these characteristics, projects in the group of sub-linear growth rate are statistically significantly larger than those in the group with linear evolution, thus falsifying hypotheses $H3''$, $H4''$, $H5''$, and $H6''$. This relationship is also shown in Figure 4 for size. For summary, Table III shows the distributions of growth patterns in different data sets and Table IV gives an overview of all hypotheses, tests, and results.

Therefore, it seems as if projects of small size with few participants and relatively equal distribution exhibit linear growth patterns, while larger ones either show sub- or super-linear growth rates, again depending on size. This leads to slight modifications on the reasonings given above: when a project grows over time, it seems to either have the potential for even further growth, may be due

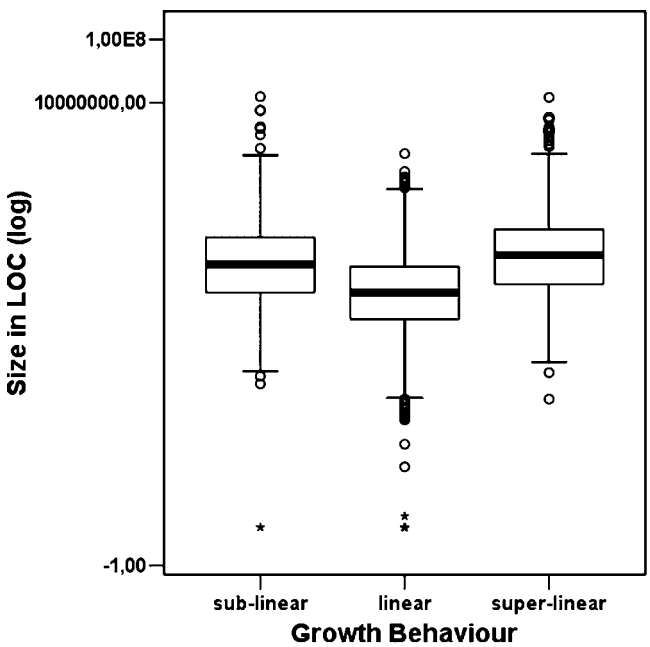

Figure 4. Boxplot of project size distribution depending on revised growth behaviour.

DOI: 10.1002/smr

Table III. Distribution of growth patterns in data sets.

Data set	N	Coefficient a of quadratic model (mean/median)	Groups	Frequency (absolute)	Frequency (relative) (%)
Full	4047	−0.504/−0.020	Super-linear	1578	39
			Other	2469	61
Mature projects	1087	−0.586/−0.011	Super-linear	425	39
			Other	662	61
Large projects	207	−0.364/0.0023	Super-linear	104	50
			Other	103	50
Full with revised groups	4047	n.a.	Super-linear	890	22
			Linear	1700	42
			Sub-linear	1457	36

to a number of possible factors including a modular initial design, attractiveness or others, and thus can achieve and sustain super-linear growth rate. On the other hand, in the absence of these factors, the laws of software evolution apply and the growth rates decrease.

4.5. Threats to validity

According to Perry *et al*. [41], threats to validity in experimental studies, under which they group version control systems as a form of retrospective artefact analysis, can be categorized into construct, internal, and external validities. Construct validity means that the independent and dependent variables accurately model the abstract hypotheses. Internal validity means that changes in the dependent variables can be safely attributed to changes in the independent variables. External validity means that the results generalize to settings outside the study.

The first, and a major source of possible threats is construct validity. Several measures used for conceptualizing different aspects for the following analyses need to be discussed in this context. The first possible issue is that LOC are used for studying the evolution of software systems instead of more traditional higher-level metrics such as number of modules or files. But, as Herraiz *et al*. [42] have shown, this approach is possible, their study validates that the evolution patterns in both cases are the same. Also, Godfrey and Tu [22] noted that the LOC grew at roughly the same rate as the number of source files in their case study of the Linux kernel. Furthermore, the definition of LOC is very much disputed [34]. In this study, we adopted the definition of our main data source, CVS [28], so comments are included. For example, Godfrey and Tu [22] have found that the percentage of comments stayed almost constant in the Linux kernel, thus, giving an extremely high correlation. This would mean that both measures can be used interchangeably, but this would merit further investigation and validation. In addition, all types of files have been included in this analysis, which might possibly also constitute a threat to validity. Due to the fact, that for example, documentation or similar files are included, the evolutionary behaviour might be affected. Robles *et al*. [25] have studied the evolution of KDE, distinguishing between several file types like code, documentation, images and others, and found that, with the exception of multimedia, the growth patterns are similar to code files. This would lead to the conclusion that results regarding evolutionary behaviour are not affected by including different file types, but these results need to be strengthened and validated.

DOI: 10.1002/smr

Table IV. Overview of hypotheses and test results.

Hypothesis	Description	Data set	Test	Sig.	Supported
$H1$	Higher number of active programmers has negative influence on productivity in mean output per programmer in lines-of-code, and, higher number of active programmers has negative influence on productivity in mean output per programmer in revisions	Complete ($N = 8621$)	Spearman correlation	Yes ($p < 0.05$)	No (coefficient extremely small)
$H2$-fullset	Quadratic growth function better fit than linear	Full ($N = 4047$)	Wilcoxon signed rank	Yes ($p < 0.01$)	Yes
$H2$-mature		Mature ($N = 1087$)	Wilcoxon signed rank	Yes ($p < 0.01$)	Yes
$H2$-large		Large ($N = 207$)	Wilcoxon signed rank	Yes ($p < 0.01$)	Yes
$H3$-fullset	Projects with super-linear growth larger in lines-of-code	Full ($N = 4047$)	Mann–Whitney U	Yes ($p < 0.01$)	Yes
$H3$-mature		Mature ($N = 1087$)	Mann–Whitney U	Yes ($p < 0.01$)	Yes
$H3$-large		Large ($N = 207$)	Mann–Whitney U	Yes ($p < 0.05$)	Yes
$H4$-fullset	Projects with super-linear growth more active in number of revisions	Full ($N = 4047$)	Mann–Whitney U	Yes ($p < 0.01$)	Yes
$H4$-mature		Mature ($N = 1087$)	Mann–Whitney U	Yes ($p < 0.01$)	Yes
$H4$-large		Large ($N = 207$)	Mann–Whitney U	Yes ($p < 0.05$)	Yes
$H5$-fullset	Projects with super-linear growth have more programmers	Full ($N = 4047$)	Mann–Whitney U	Yes ($p < 0.01$)	Yes
$H5$-mature		Mature ($N = 1087$)	Mann–Whitney U	Yes ($p < 0.01$)	Yes
$H5$-large		Large ($N = 207$)	Mann–Whitney U	Yes ($p < 0.01$)	Yes
$H6$-fullset	Projects with super-linear growth have higher inequality	Full ($N = 4047$)	Mann–Whitney U	Yes ($p < 0.05$)	Yes
$H6$-mature		Mature ($N = 1087$)	Mann–Whitney U	No	No
$H6$-large		Large ($N = 207$)	Mann–Whitney U	Yes ($p < 0.05$)	Yes
$H3'$	Projects with super-linear growth larger in lines-of-code than those with linear, and larger than those with sub-linear	Full revision ($N = 4047$)	Mann–Whitney U	Yes ($p < 0.01$)	Yes

DOI: 10.1002/smr

Table IV. *Continued.*

Hypothesis	Description	Data set	Test	Sig.	Supported
$H3''$	Projects with linear growth larger in lines-of-code than those with sub-linear	Full revision ($N = 4047$)	Mann–Whitney U	No	No
$H4'$	Projects with super-linear growth more active in number of revisions than those with linear, and more active than those with sub-linear	Full revision ($N = 4047$)	Mann–Whitney U	Yes ($p < 0.01$)	Yes
$H4''$	Projects with linear growth more active in number of revisions than those with sub-linear	Full revision ($N = 4047$)	Mann–Whitney U	No	No
$H5'$	Projects with super-linear growth have more programmers than those with linear, and more programmers than those with sub-linear	Full revision ($N = 4047$)	Mann–Whitney U	Yes ($p < 0.01$)	Yes
$H5''$	Projects with linear growth have more programmers than those with sub-linear	Full revision ($N = 4047$)	Mann–Whitney U	No	No
$H6'$	Projects with super-linear growth have higher inequality than those with linear, and higher inequality than those with sub-linear	Full revision ($N = 4047$)	Mann–Whitney U	Yes ($p < 0.01$)/No	Partly
$H6''$	Projects with linear growth have higher inequality than those with sub-linear	Full revision ($N = 4047$)	Mann–Whitney U	No	No

DOI: 10.1002/smr

The next construct to be discussed is the notion of programmer, which, although nearly no results on personal level are given in this study, is important as it forms the basis for $H3$, and indirectly for $H4$. A programmer in these analyses is defined by counting those people committing source code changes through their CVS account, thus only people with such accounts are measured. In some projects, people could be contributing code without CVS account, which sometimes is only granted to long-time participants, by sending it to one of those persons who then does the actual commit. Therefore, the number of programmers might actually be higher than the number reported here. This fact is very problematic to check. In general, there are several possibilities of attributing authorship of source code to persons, which are to use the associated CVS account (as done here), to mine the versioning system comments for any additional attributions, to infer from attributions in the source code itself, or by questionnaires or intimate knowledge of a project and its participants. Attributions in source code or commit comments are highly dependent on existence and form of a project's standards, and therefore are also difficult to implement for larger data sets. Ghosh and Prakash [14] have implemented a solution based on source code attributions for a set of over 3000 projects, with about 8.4% of the code base remaining uncredited, and with the top authors containing organizations like the Free Software Foundation or Sun Microsystems. Nevertheless, they have found a similar distribution of participation as found in this study's data set, as have most other approaches like questionnaires [36] or case studies of larger projects [6–9]. In a case study of the OpenACS project under participation of project insiders and using the strict standards for CVS comments, Demetriou *et al.* [43] have found that only 1.6% of revisions pertained to code committed for someone without CVS privilege. Lastly, the notion of inequality or concentration is quite difficult, as there are numerous measures for this concept, including Gini, Atkinson, Theil or Herfindahl measures, each of which has advantages and disadvantages, for example, the Gini measure has a maximum lower than 1 depending on the number of individuals. In this study, a very simple measure was used as described, computing for each programmer in the team the squared difference between the percentage of revisions he contributed, and the percentage he should contribute if everyone contributed the same amount (excluding single-person projects), and summing up for all team members. Preliminary analyses have shown that this measure is highly correlated with others like the Gini coefficient, but further studies should be undertaken to strengthen this.

Also, Howison and Crowston [44] give an overview of problems associated with mining data from Sourceforge.net, but as these mostly concern published data from the website, which is a less concern for this study as it mostly uses the CVS archives accessed directly, these points do not apply.

Internal validity seems to be a comparably lesser problem in this study. All precautions have been taken to avoid any possible errors in establishing relationships between the variables, including rigorous testing using appropriate methods and employing strict significance limits with Bonferroni correction where necessary.

Of more importance and relevance is a discussion of possible threats to external validity, i.e., to the generalizability of the results. This mostly hinges on whether the selected data set is a good sample of the overall population, i.e., whether Sourceforge.net is representative for the open source ecology. Due to the de-centralized nature of open source, this population and its characteristics are largely unknown. For example, the number of existing open source projects cannot be known, summing up all projects in different repositories like Sourceforge.net and others, and known projects with their own web presence could only give a lower bound for this. Naturally, the same is true for the structure of this population, e.g., the distribution of project sizes, or even for a more elaborate concept as

DOI: 10.1002/smr

success of a project [30,45–47]. For the case of this study, the characteristics of the data set used are in several key features, both on project level and within projects, similar to other data sets and projects studied. As mentioned before, the inequality in participation within projects was already reported in many other works, using different approaches. Also, the distribution of size and number of participants between projects are similar to other studies [11,12,14]. We have also tried to counter the problem, which on the other hand seems to be a feature of the overall population, of many small projects in the data set by separately examining the subset of large projects. Nevertheless, more research is necessary to be able to determine to what extent Sourceforge.net is representative for the overall open source project population.

For further validation of the results of this study, we propose to perform similar analyses based on richer, larger, and newer data set, i.e., also covering the time passed since data collection. Currently, several efforts are under way to ease the collection of and access to this sort of data, which were not available before. For example, the FLOSSMOLE project (http://ossmole.sourceforge.net) offers access to data from Sourceforge.net, but also now Freshmeat, Rubyforge, and Objectweb, limited to data published on the web pages, not the source code versioning system. The Libresoft web page at Universidad Rey Juan Carlos (http://libresoft.urjc.es) offers data and analyses results on Sourceforge.net and many major and well-known projects mostly based on source code versioning systems. Also, the newly founded and EC-funded FLOSSMETRICS project (http://www.flossmetrics.org) aims at providing access to both raw data and analyses of a huge number of projects using diverse approaches. Overall, the public availability of large and rich data sets from open source projects will enable studies in the field of software engineering to be done with more confidence than ever before, and to be validated, replicated, and enhanced much more easily and in a more open fashion.

5. CONCLUSIONS

In this paper, the evolution of a large sample of open source software systems has been analysed. The evolution of commercial systems has been an issue that has long been a centre of research, and therefore a coherent theoretical framework of software evolution has been developed and empirically tested. The data collection methodology relying on a large software repository and the respective source code control systems has been described, and an overview on the collected data on several thousand projects was given. Several approaches including different subsets of the data were employed to analyse aspects of growth behaviour, with a quadratic model being shown as better suited to model the behaviour than a linear one. The most interesting fact is that while in the mean the growth rate is linear or decreasing over time according to the laws of software evolution, a significant percentage of projects is able to sustain super-linear growth. There is a positive relationship between the size of a project, the number of participants, and the inequality in the distribution of work within the development team with the presence of super-linear growth patterns. On the other hand, there is evidence for a group of projects of moderate size which shows decreasing growth rates, while small projects in general exhibit linear growth. A resulting hypothesis, therefore, is that there are factors which during the growth of a project either allow to evade the laws of software evolution and sustain super-linear growth, or lead to the project succumbing to them and following a decreasing growth pattern. A possible explanation for this fact is that projects with a super-linear growth rate are able to implement a certain organizational model, the chief programmer team. This form, present in open source software development, through measures like strict modularization and

DOI: 10.1002/smr

self-selection for tasks seems to be able to at least delay the negative effects arising during evolution. In addition, especially the fourth law of software evolution, 'conservation of organizational stability' [18], implying constant incremental effort, might be violated especially in very large and prominent projects which attract an ever increasing number of participants.

REFERENCES

1. Raymond ES. *The Cathedral and the Bazaar*. O'Reilly & Associates: Cambridge MA, 1999.
2. Perens B. The open source definition. *Open Sources*: *Voices from the Open Source Revolution*, DiBona C *et al* (eds.). O'Reilly & Associates: Cambridge MA, 1999.
3. Stallman RM. *Free Software*, *Free Society*: *Selected Essays of Richard M. Stallman*. GNU Press: Boston, MA, 2002.
4. McConnell S. Open-source methodology: Ready for prime time? *IEEE Software* 1999; **16**(4):6–8.
5. Vixie P. Software engineering. *Open Sources*: *Voices from the Open Source Revolution*, DiBona C *et al* (eds.). O'Reilly & Associates: Cambridge MA, 1999.
6. Mockus A, Fielding R, Herbsleb J. A case study of open source software development: The Apache server. *Proceedings 22nd International Conference on Software Engineering*, 2000; 263–272.
7. Mockus A, Fielding R, Herbsleb J. Two case studies of open source software development: Apache and Mozilla. *ACM Transactions on Software Engineering and Methodology* 2002; **11**(3):309–346.
8. Koch S, Schneider G. Effort, cooperation and coordination in an open source software project: GNOME. *Information Systems Journal* 2002; **12**(1):27–42.
9. Dinh-Trong TT, Bieman JM. The FreeBSD project: a replication case study of open source development. *IEEE Transactions on Software Engineering* 2005; **31**(6):481–494.
10. Crowston K, Scozzi B. Open source software projects as virtual organizations: Competency rallying for software development. *IEE Proceedings—Software Engineering* 2002; **149**(1):3–17.
11. Hunt F, Johnson P. On the pareto distribution of sourceforge projects. *Proceedings Open Source Software Development Workshop*, 2002; 122–129.
12. Krishnamurthy S. Cave or community? An empirical investigation of 100 mature open source projects. *First Monday* 2002; **7**(6). Available at: http://www.firstmonday.org/issues/issue7_6/krishnamurthy/.
13. Dempsey BJ, Weiss D, Jones P, Greenberg J. Who is an open source software developer? *Communications of the ACM* 2002; **45**(2):67–72.
14. Ghosh R, Prakash VV. The orbiten free software survey. *First Monday* 2000; **5**(7). Available at: http://www.firstmonday.org/issues/issue5_7/ghosh/.
15. Koch S. Profiling an open source project ecology and its programmers. *Electronic Markets* 2004; **14**(2):77–88.
16. Belady LA, Lehman MM. A model of large program development. *IBM Systems Journal* 1976; **15**(3):225–252.
17. Lehman MM, Belady LA. *Program Evolution—Processes of Software Change*. Academic Press: London, 1985.
18. Lehman MM, Ramil JF. Rules and tools for software evolution planning and management. *Annals of Software Engineering* 2001; **11**:15–44.
19. Turski WM. Reference model for smooth growth of software systems. *IEEE Transactions on Software Engineering* 1996; **22**(8):599–600.
20. Kemerer CF, Slaughter S. An empirical approach to studying software evolution. *IEEE Transactions on Software Engineering* 1999; **25**(4):493–509.
21. Scacchi W. Understanding free/open source software evolution. *Software Evolution*, Madhavji NH, Lehman MM, Ramil JF, Perry D (eds.). Wiley: New York NY, 2004.
22. Godfrey MW, Tu Q. Evolution in open source software: A case study. *Proceedings International Conference on Software Maintenance*, 2000; 131–142.
23. Paulson JW, Succi G, Eberlein A. An empirical study of open-source and closed-source software products. *IEEE Transactions on Software Engineering* 2004; **30**(4):246–256.
24. Robles G, Amor JJ, Gonzalez-Barahona JM, Herraiz I. Evolution and growth in large libre software projects. *Proceedings International Workshop on Principles of Software Evolution* (*IWPSE 2005*), 2005.
25. Robles G, Gonzalez-Barahona JM, Merelo JJ. Beyond source code: The importance of other artifacts in software development (a case study). *Journal of Systems and Software* 2006; **79**(9):1233–1248.
26. Capiluppi A, Lago P, Morisio M. Evidences in the evolution of OS projects through Changelog analyses. *Proceedings 3rd Workshop on Open Source Software Engineering*, *25th International Conference on Software Engineering*, 2003; 19–24.
27. Robles-Martinez G, Gonzalez-Barahona JM, Centeno-Gonzalez J, Matellan-Olivera V, Rodero-Merino L. Studying the evolution of libre software projects using publicly available data. *Proceedings 3rd Workshop on Open Source Software Engineering*, *25th International Conference on Software Engineering*, 2003; 111–116.

DOI: 10.1002/smr

28. Fogel K. *Open Source Development with CVS*. Coriolis Open Press: Scottsdale AZ, 1999.
29. Nakakoji K, Yamamoto Y, Nishinaka Y, Kishida K, Ye Y. Evolution patterns of open-source software systems and communities. *Proceedings International Workshop on Principles of Software Evolution*, 2002.
30. Stewart KJ. OSS project success: From internal dynamics to external impact. *Collaboration*, *Conflict and Control*: *The 4th Workshop on Open Source Software Engineering* (*ICSE 2004*), 2004.
31. Cook JE, Votta LG, Wolf AL. Cost-effective analysis of in-place software processes. *IEEE Transactions on Software Engineering* 1998; **24**(8):650–663.
32. Atkins D, Ball T, Graves T, Mockus A. Using version control data to evaluate the impact of software tools. *Proceedings 21st International Conference on Software Engineering*, 1999; 324–333.
33. Fischer M, Pinzger M, Gall H. Populating a release history database from version control and bug tracking systems. *Proceedings 19th IEEE International Conference on Software Maintenance* (*ICSM'03*), 2003; 23–32.
34. Park RE. Software size measurement: A framework for counting source statements. *Technical Report CMU/SEI-92-TR-20*, Software Engineering Institute, Carnegie Mellon University, Pittsburgh PA, 1992.
35. Hahsler M, Koch S. Discussion of a large-scale open source data collection methodology. *Proceedings Hawaii International Conference on System Sciences* (*HICSS-38*), 2005.
36. Hertel G, Niedner S, Hermann S. Motivation of software developers in open source projects: An internet-based survey of contributors to the Linux kernel. *Research Policy* 2003; **32**(7):1159–1177.
37. Robles G, Koch S, González-Barahona JM. Remote analysis and measurement of libre software systems by means of the CVSAnalY tool. *Proceedings 2nd ICSE Workshop on Remote Analysis and Measurement of Software Systems*, *26th International Conference on Software Engineering*, 2004.
38. Brooks FP Jr. *The Mythical Man-Month*: *Essays on Software Engineering* (Anniversary edn). Addison-Wesley: Reading MA, 1995.
39. Mills HD. Chief programmer teams: Principles and procedures. *Report FSC 71-5108*, IBM Federal Systems Division, Gaithersburg MD, 1971.
40. Baker FT. Chief programmer team management of production programming. *IBM Systems Journal* 1972; **11**(1):56–73.
41. Perry DE, Porter AA, Votta LG. Empirical studies of software engineering: A roadmap. *The Future of Software Engineering*, Finkelstein A (ed.). ACM Press: New York NY, 2000.
42. Herraiz I, Robles G, González-Barahona JM, Capiluppi A, Ramil JF. Comparison between SLOCs and number of files as size metrics for software evolution analysis. *Proceedings 10th European Conference on Software Maintenance and Reengineering*, 2006.
43. Demetriou N, Koch S, Neumann G. The development of the OpenACS community. *Open Source for Knowledge and Learning Management*: *Strategies Beyond Tools*, Lytras M, Naeve A (eds.). Idea Group: Hershey PA, 2006.
44. Howison J, Crowston K. The perils and pitfalls of mining SourceForge. *Proceedings International Workshop on Mining Software Repositories*, 2004; 7–11.
45. Crowston K, Annabi H, Howison J. Defining open source software project success. *Proceedings ICIS 2003*, 2003.
46. Crowston K, Annabi H, Howison J, Masango C. Towards a portfolio of FLOSS project success measures. *Collaboration*, *Conflict and Control*: *The 4th Workshop on Open Source Software Engineering* (*ICSE 2004*), 2004.
47. Weiss D. Measuring success of open source projects using web search engines. *Proceedings 1st International Conference on Open Source Systems*, 2005; 93–99.

AUTHOR'S BIOGRAPHY

Stefan Koch is an Associate Professor of Information Business at the Vienna University of Economics and Business Administration. He received a MBA in Management Information Systems from Vienna University and Vienna Technical University, and a PhD from Vienna University of Economics and Business Administration. Currently, he is involved in the undergraduate and graduate teaching programme, especially in software project management and ERP packages. His research interests include cost estimation for software projects, the open source development model, software process improvement, the evaluation of benefits from information systems and ERP systems. He has edited a book titled 'Free/Open Source Software Development' for an international publisher in 2004, and has published extensively on analysis, quantification and effort estimation for open source projects.

DOI: 10.1002/smr